RFreeStem: A multilanguage rule-free stemmer

Xavier Baril [1], **Oihana Coustié** [2], **Josiane Mothe** [3], **Olivier Teste** [4]

1. *Airbus Opération SAS,*
   316 route de Bayonne, Boite postale interne M3014, 31060 Toulouse cedex, France
   xavier.baril@airbus.com
2. *Airbus Opération SAS,*
   316 route de Bayonne, Boite postale interne M3014, 31060 Toulouse cedex, France
   oihana.coustie@irit.fr
3. *IRIT*
   118, Route de Narbonne, 31062 Toulouse cedex 04, France
   josiane.mothe@irit.fr
4. *IRIT*
   118, Route de Narbonne, 31062 Toulouse cedex 04, France
   olivier.teste@irit.fr

ABSTRACT. *With the large expansion of available textual data, text mining has become of special interest. Due to their unstructured nature, such data require important preprocessing steps. Among them, stemming is a popularly used preprocessing method that extracts the root of the words. However, the most popular algorithms are based on the application of rules, and therefore highly language-related. We propose a new approach, the RFreeStem, that is rather based on corpus and can therefore be applied on many languages.*

RÉSUMÉ. *La grande disponibilité de données textuelles a rendu populaire les recherches de fouille de texte. Parmi les importants prétraitements nécessaires, la racinisation s'est imposé comme une étape incontournable. Pourtant, les algorithmes les plus utilisés sont basés sur l'application successives de règles. Cette construction les rend fortement dépendants de la langue d'application. Nous proposons ici une nouvelle approche, le RFreeStem, qui se base sur le corpus étudié et promet de pouvoir être appliqué à plusieurs langues.*

KEYWORDS: *Text mining, information retrieval, Sentiment analysis, Stemmer, NLP*

MOTS-CLÉS : *Fouille de texte, recherche d'information, Analyse de sentiments, racinisation*

1. Introduction

In the past decades, the overwhelming progress in technologies and Web researches has lead to a high increase of available machine readable documents (Moral *et al*., 2014). Such unstructured data represent a huge potential of information to retrieve. Yet, the absence of structure, along with their high dimensionality make them difficult to process, with an important amount of necessary pre-treatment. Among the numerous pre-processing tasks for text analysis, the idea of stemming the words in the textual data has emerged since the late sixties (Lovins, 1968) and is now almost systematically used (Marcos-Pablos, García-Peñalvo, 2019).

Stemming can be described as the process of replacing a word by its morphological root — the *stem*. Such an algorithm is referred to as a *stemmer*. The stem can be a sub-string or a concatenation of sub-strings of the word, or even a modified sub-string, as in (Porter, 1980), who stems *happy* to *happi*. More than the obtain stem itself, the main result of a stemmer stands in the groups created by regrouping words identically stemmed. Such groups are often referred to as *conflation* groups (Frakes, Baeza-Yates, 1992). For example, (Porter, 1980) would conflate *happy* and *happier*, both stemmed to *happi*. Having words stemmed together automatically reduces the size of the vocabulary since it removes the morphological variants of the words of a lexicon (Jivani *et al*., 2011).

The assumption made here is that morphologically related words have similar meanings and represent the same idea. However, this hypothesis is questionable, that is why many research papers decided to work on lemmatizing algorithms instead. (Moral *et al*., 2014) describes *lemmatization* as linguistic-based approach that relies on Part of Speech(PoS) and context. They are less dependants on the morphological variants of the words than stemmers are. To do so, they often draws on dictionaries or thesauri. Even if they offer prospectively more accurate results, their high complexity make them unpopular. Lemmatization will therefore not be detailed furthermore in this paper, since it is another field of research. We will rather focus on stemmers, reputed to be faster and more popular (Jivani *et al*., 2011).

Indeed, stemming has become an almost inescapable step in some text mining tasks. Among them, *Information Retrieval* (IR) algorithms automatically analyze documents to extract information to answer user queries (Hull, 1996): since the stem represents a broader concept than the word, more documents are expected to match the query. Other text mining fields extensively use stemming pre-processing. (Jivani *et al*., 2011) describes it as a requirement for many *Natural Language Processing* (NLP) applications, including text clustering and categorization or sentiment analysis — the process to automatically describe the sentiments of the author of a message.

However, the most widely used stemmer are based on rules to be applied on the words, in order to remove irrelevant parts. These rules rely on the morphological forms of the words and thus are necessarily language-specific. Whereas highly-spoken languages have their rules well described, it is difficult to find an implementation of those algorithms for rare languages or dialects. Moreover, the number of rules, as well

as the quality of the stemmer, seem to highly depend on the language it is applied on (Kraaij, Pohlmann, 1996). To be more specific, linguistics distinguish two different types of language structures (Moral *et al*., 2014):

– Analytic (or isolating) languages, with little morphology structures, where the variants are more likely to be expressed by helper words than by word variations (e.g. Chinese, Vietnamese, modern English).

– Synthetic languages, with strong morphology structures. Among them we find: (i) The inflective language, like Latin languages (French, Italian) or Germanic languages (German, Dutch, Swedish, (Braschler, Ripplinger, 2004)). They have numerous prefixes and suffixes applied to modify the grammatical nature or even the meaning of the words.(ii) The agglutinative language, like Finnish, Japanese or Basque. In those languages the suffixes do not only represent inflections but can also be the concatenation of other morphemes. For example, in basque, *exte* means *house* and *etxean* means *in the house*.

Regarding those different categories of languages, (Jivani *et al*., 2011) affirms that stemming has proven more efficient on languages with complex morphology, that is to say on synthetic languages. (Braschler, Ripplinger, 2004) come to the same conclusion with its study on Germanic ones. Yet, on such languages, a great number of rules would be needed, in order to remove all the affixes, which would make the stemmer poorly flexible.

Furthermore, stemming evaluation can be performed by comparing its results to a set of validated labels. In this case, further than the traditional precision and recall, (Paice, 1994) defines the *overstemming* and *understemming* errors. Overstemming error counts the number of words stemmed together whereas they shouldn't have and understemming error census the words with different stems that should have been stemmed together. Paice proposes the calculation of indicators based on such metrics. However, they all assume the access to an objective label set, which is difficult to census in practice, especially on rare languages.

In the light of these observations, our contribution consists in a rule-free stemmer, that can be applied on multiple languages. To cope with the previously mentioned evaluation issues, we will evaluate our stemmer by measuring its impact on the results of data mining tasks. We will also assess what we call *compression power* — the ability of a stemmer to compress the size of a vocabulary. We will measure it thanks to the famous *Index Compression Factor* (Sirsat *et al*., 2013), that asses the strength of stemmers. Finally, we shall be paying special attention to the stem produced. Although (Jivani *et al*., 2011) mentions that stemmers do not need to be words from the language, it can nevertheless be crucial that the stems are humanly readable in tasks like query expansion or dictionary look-up (Hull, 1996).

In accordance with those matters, we first propose to census the different types of stemming methods found in the literature in section 2, before describing ours, the RFreeStem in setion 3. We will finally describe, in section 4 an evaluation framework to assess the results of our stemmer on different text mining tasks and languages.

2. Related work

The literature offers an important variety of stemming algorithms that can mostly be categorized in two types of approach : (a) the rule-based methods, (b) the corpus-based methods. The following subsections propose to census some of the most important studies on stemming algorithms in those two categories.

2.1. Rule-based methods

Most of the well-known stemming methods are based on the application of rules. Among them, *affix stripping* gathers procedures to remove the suffixes and, sometimes, the prefixes of the words. In 1968, (Lovins, 1968) proposed a first suffix stripping method. Aggressive and fast, the algorithm matches the end of the word with the longest element of a suffix list, and applies rules to modify the matching end. Lovins's algorithm is reputed to be unreliable since its list of suffixes is based on a very technical-oriented vocabulary. Yet, technical and scientific words tend to derive from Latin languages and therefore to be highly inflectional, whereas common modern English is rather isolating[1].

The now most popular algorithm was soon proposed by (Porter, 1980). With 5 successive steps, it is a bit slower than its 2-step predecessor, but has experimentally proven more accurate (Willett, 2006), (Paice, 1996). Initially, Porter exclusively defined those successive rules to stem English words. Yet, its popularity motivated the implementation of `snowball`(Porter, 2001), a detailed framework that enables users to implement their own version of Porter stemmer in any language. However, it remains a current subject of study to find such implementations for less common or highly inflectional languages (e.g. Bahasa Indonesian (Tala, 2003), Dutch (Kraaij, Pohlmann, 1994)).

Many other rule-based methods were proposed afterwards. Among them, the light *S stemming* only deals with plural forms, and so presents little compression power. On the other hand, (Paice, 1990) proposed a strong algorithm that applies successive deletion and replacement rules. (Jivani *et al.*, 2011) argues that this method often presents a high over-stemming error. (Dawson, 1974) implemented an adaptation of Lovins method with much more suffixes and rules, which unfortunately makes the algorithm hardly reusable (Jivani *et al.*, 2011). (Krovetz, 1993) proposes an approach based on inflections and derivations, with its Krovetz stemmer, *KSTEM*. It relies on an inflection-free lexicon to withdraw inflections, before analyzing the derivations — variants that change the grammatical nature of words. The author acknowledges that the performance strongly depends on the chosen lexicon. We even add that finding such an exhaustive lexicon might not be feasible for rare language or dialect.

1. English language is actually slightly inflectional, due to its heritage of old English. Thus, modern English is more fit for stemming than purely isolating languages like Mandarin or Indonesian (Moral *et al.*, 2014).

Those rule-based methods are undeniably the most recognized and used in information retrieval, thanks to their fast computation skills and easy implementations (Harman, 1991). Nevertheless, adapting those methods to highly-inflectional languages would lead to a large number of necessary rules. Consequently, recent papers that focus on stemming text from those languages are inclined to look for other approaches than rule-based ones.

2.2. Corpus-based methods

In order to cope with the strong language dependence of rule-based stemming techniques, some corpus-based methods were proposed. Most of them rely on statistical principles. (Hafer, Weiss, 1974) developed a *Successor variety* stemmer which cuts the words in two parts : if the first part belongs to the corpus, this first part becomes the stem. The cutting algorithm is based on the evolution, within the word's letters, of the *successor variety* : the number of distinct characters that follow a string within all the words of the corpus. Thus, the resulting conflations strongly depend on the corpus, and yet, systematically choosing the first part of the word as a stem seems to be a language-dependant approach.

Another method, — the *ngram stemmer*— proposed by (Adamson, Boreham, 1974), uses a common string clustering method to create conflation groups : a hierarchical single linkage clustering of the words. To evaluate the word pairwise distance, the authors used the Dice distance, corresponding to the number of distinct shared digrams — sequence of two consecutive letters. Using an unsupervised clustering method to create conflation groups makes the stemming corpus-dependant but also offers a very flexible way to manage stemming strength. However, single linkage clustering is known for its important algorithm complexity, due to the creation of a quadratic distance matrix. Very similarly, *YASS* (Yet Another Stripping Stemmer), described in (Majumder *et al*., 2007), clusters words with linkage hierarchical methods, but has also implemented word distance metrics that encourage long suffix matching. According to (Jivani *et al*., 2011), they make the method more adapted for languages that are richly suffixed.

As a last interesting example, (Soricut, Och, 2015) proposed an unsupervised corpus-based method that automatically identifies meaningful rules to strip prefixes, with overall good results. The unsupervised learning on the corpus makes this method highly flexible and potentially adaptable to many languages.

From our reading of the literature, some stemming features appear to be interesting and have constituted the baseline of the method we propose. A rule-free method offers the possibility for the stemmer to be applied on many languages. Moreover, an unsupervised hierarchical clustering method does not need any prior knowledge and enables a high flexibility in the stemmer strength. Finally, the use of ngrams seems particularly fit for strings classification. The following section describes how those features were implemented for the RFreeStem algorithm.

3. The RFreeStemer: a new stemmer

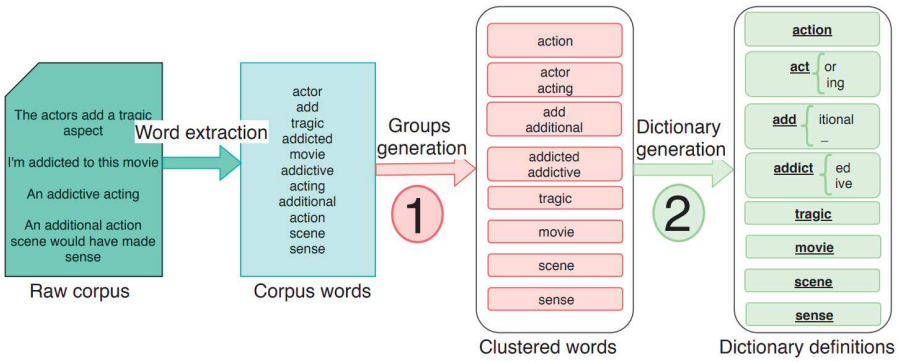

Figure 1. The general approach of RFreeStem stemmer. First step (1) is presented in section 3.1. Second step (2) is detailed in section 3.2

To answer the detected issues of existing stemmers, we propose a new approach with RFreeStem stemmer. Our corpus-based algorithm clusters the words of a corpus in order to create conflation groups. It is therefore an unsupervised learning method, that does not need any external knowledge. The proposed clustering is hierarchical — which allows flexibility in term of stemmer strength — and based on ngram similarity. Moreover, the clustering method promises to be much faster than any distance-matrix-based clustering. Indeed, instead of creating such a quadratic matrix, the algorithm only runs through the ngrams once. The RFreeStem stemmer is divided in two steps: after extracting all the words from the corpus, the group generation step clusters the words into conflation groups, and the dictionary generation step creates a readable dictionary out of the stemmed groups. Those steps are resumed in Figure 1 and describes in the following subsections.

3.1. Word clustering

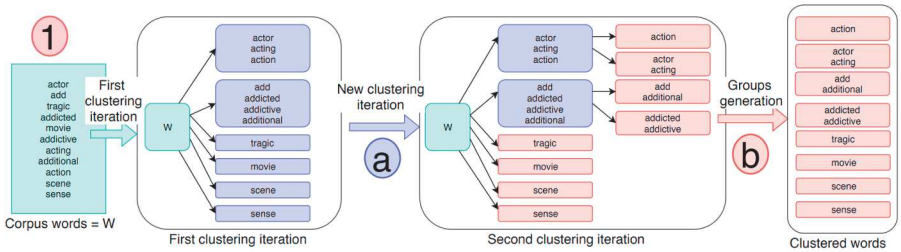

Figure 2. The clustering method (1) with (a) the recursive division (see 3.1.1) and (b) the dendrogram cutting (see 3.1.3

The first step of our process consists in clustering the corpus words. After the extraction of all the words of the corpus, they are processed to be regrouped by con-

flation. To do so, we use a hierarchical clustering method — generally more adapted for text clustering— with a divisive approach : starting from the whole set of words, we recursively split it into clusters, as illustrated in Figure 2. We call those recursive steps *divisive steps*, and describe them in the subsection 3.1.1. We therefore discuss the choice of the number of divisive step needed in section 3.1.3. Indeed, deciding how to cut the dendrogram directly impacts the conflation groups and the stemmer strength.

3.1.1. Divisive step algorithm

At each divisive step, a set of several words is divided into several groups. The algorithm 1 formally describes this process.

Algorithm 1: Divisive step algorithm

```
   input  : A set of (unique) words W, an integer n;                    // n as in ngram
   output: A set of word groups
 1 ngram.set ← findnGram(words = W, n = n);
 2 ngram.set ← scoreFunction(set = ngram.set);
 3 ngram ← first(ngram.set);
 4 G ← emptyGroup();
 5 while ngram != last(ngram) and size(W) > 0 do
 6 |   matching.words = grep(pattern = ngram, set = W);
 7 |   if 0 < size(matching.words) < size(W) then
 8 |   |   remove(elementsToRemove = matching.words, set = W);
 9 |   |   g ← createGroup(elements = matching.words);
10 |   |   addGroup(element = g, group = G);
11 |   end
12 |   ngram = next(ngram);
13 end
14 return G;
```

The algorithm uses the functions : (i) `findnGram` that extracts all the unique ngrams within a set of words (ii) `scoreFunction` that sorts a set of ngram and returns it. The implementation of that function is described in subsection 3.1.2. (iii) `first`, `last` and `next` which respectively returns the first, last or next element of a set (iv) `createGroup` that creates a group with words (v) `emptyGroup` and `addGroup` which respectively initializes a empty set of clusters and a cluster to a set (vi) `grep` which returns the words that match a pattern.

In the first divisive step, the input set of words is the whole corpus, whereas in each intermediate ones, the input is an already formed cluster. For an input W, $w \in W$ is a word of the set, on an alphabet $\mathcal{A}$. If $w = a_1 a_2 .. a_k$, where k is the length of w and $\forall i, a_i \in \mathcal{A}$, then a ngram of w is a subset $a_i .. a_{i+n}$, $i \in [\![1; k-n]\!]$. Notice that w exactly has $k - n + 1$ ngrams. We propose to extract all the distinct ngrams of all the words. Let $\mathcal{N}$ be such a set of ngrams. Our method then sorts the ngrams (subsection 3.1.2), which are then browsed exactly once. For each $ng \in \mathcal{N}$, all the words that contain ng are clustered together. This one-browse clustering offers an interesting decrease in computational execution time, compared with traditional distance-based clustering algorithms. In addition, the condition $\texttt{size}(W) = 0$ — all the words are clustered — is generally reached before the end of the browse, stopping the execution.

3.1.2. Scoring function choice

It appears from the previously described algorithm that the definition of the function that sorts the ngram is crucial — We call it the *scoring function*. Indeed, it is designed to give a score to each ngram, knowing they will be browsed according to these scores. A ngram with a high score will be responsible for the clustering of all the words that contain this ngram together. We define the *generated group* of an ngram as the set of words that contain this ngram. Regarding our algorithm, the generated groups of well-scored ngrams will become clusters, and the final generated groups will gather words that will be stemmed together. Hence, if the ngram *tion* has a better score than *clas*, then *classification* will be clustered with the *tion*-words — like *imitation*, *attention* — instead of being united with words like *class* or *classify*. To avoid this situation, we discuss interesting features that can define a "good" ngram:

– The number of words containing this ngram, which also is the size of the generated group. A small value only groups a few words together, which conflicts with the divisive approach of our hierarchical clustering. On the other hand, the biggest-size ngrams — the most frequent ones — are often common affixes (e.g. *tion*). A middle ground size could be the mean size of all the generated groups.

– The homogeneity of the generated group, that can be evaluated with: (i) A string distance within the generated group (e.g. edit distance) (ii) The number of ngrams in the generated group. An homogeneous group is expected to have few ngrams. (iii) A ngram distance on those ngrams, like Dice distance (vi) The number of common letters within the generated group words. This idea relates to the further dictionary creation we propose : a high number of common letters means a longer stem.

– The heterogeneity of the generated group. One measure could be the number of other generated groups that contain the ngram. Indeed, we easily imagine that *tion* is likely to appear in almost every natural conflation group. However, the relation "ng_1 belongs to the generated group of ng_2" is reciprocal. Therefore, this quantity is precisely the number of ngrams in the generated group of ng_1 (see homogeneity).

– Another intuition while refining this algorithm on English datasets was to penalize ngrams that were likely to be affixes — ngrams whose positions in the words are often either at the beginning or at the end. We feel however that this criteria is highly language-related and thus, in conflict with our rule-free principle.

We finally propose to score the ngrams with the following function :

$$s(ng_i) = \frac{1}{2} \times (\texttt{mean}(\texttt{DiceDistance}(\mathcal{N}_i)) + ||\mathcal{N}_i| - id.size|)$$

where $\mathcal{N}_i \subset \mathcal{N}$ is the set of ngrams in ng_i's generated group, $id.size$ is the ideal size of a group, namely the mean size of all the generated groups and `DiceDistance` function returns all the pairwise Dice distance for a set of ngrams. The first member represents the group homogeneity : a low Dice distance between the ngrams of a generated group reflects a group with similar words. The second member penalizes the gap between the ideal and the real size of a generated group. Based on several

experiments, we are confident that this scoring function will grant a good score to the ngrams reflecting the real stem of words.

3.1.3. Cutting the dendrogram into clusters

Once the dendrogram is generated, a valuable question is to know how to deduce the groups. Indeed the example of figure 2 shows three possible variants : the example chooses to stop after the second iteration. Though, we could have stopped at the first one, or continue to split another time. We define the *depth* of the clustering as the chosen number of iterations. A high depth value leads to a very light stemming, with many words stemmed alone or in small groups. This configuration will decrease the risk of overstemming. On the contrary, a low value of depth leads to large and fewer groups. Such an aggressive stemming decreases the understemming error rate, and increases the compression power of the stemmer. Since we think that depth value is highly data-dependent, we do not fix it, and will rather compare the experimental results obtained for different values of depth.

For this first version of the RFreeStem stemmer, we only consider homogeneous value of depth : the clustering process stops at the same iteration for each group. Though, it could have been more accurate to allow groups to have different depths. This approach would have required the definition and implementation of a stopping criteria, to automatically decide when a cluster is considered homogeneous enough. We let that interesting question for a future work extension.

3.2. Dictionary generation

The second step of our algorithm aims to generate a dictionary out of the groups previously created. In each homogeneous group $\mathcal{W}_g$, this step consists in finding the fix and variable parts in the words of the conflation group. A fix part is defined as a sequence of consecutive letters that can be found in each word of the group. For instance, the group formed by the two words *classification* and *unclarified* has two fix parts : `cla` and `ifi`. The variable parts are simply the non-fix parts of the words. In the example, we would respectively find {`ss`, `cation`} and {`un`, `r`, `ed`}. The goal is to obtain a dictionary censusing the stems of the group (fix parts) and all the possible values taken by the variable parts, as shown in the last part of Figure 1. In the minimal example of the words *classification* and *unclarified*, we actually want to align the matching variable parts, to obtain :

$$\left\{ \begin{array}{l} \text{un} \\ \_ \end{array} \right. \text{cla} \left\{ \begin{array}{l} \text{r} \\ \text{ss} \end{array} \right. \text{ifi} \left\{ \begin{array}{l} \text{ed} \\ \text{cation} \end{array} \right.$$

Yet, the number of variable is not constant. In order to be able to align them, we refine our definition : a variable part v_{i_j} is an ngram between the i^{th} and $i + 1^{th}$ fix parts, with $i \in [\![0; F]\!]$, where F is the number of fix parts in a group, and $j \in [\![1; |\mathcal{W}_g|]\!]$. For instance here, we have $v_{1_1} =$ `r` and $v_{1_2} =$ `ss`. For a given i, the vector $v_i = \{v_{i_j}, j \in [\![1; |\mathcal{W}_g|]\!]\}$ is the set of all the value taken by the variable parts.

To illustrate the difficulty of finding the fix parts, we extend our example to $\mathcal{W}_g = \{$ *classification*, *declaration*, *unclarified*, *clarity*, *claimed*, *cleansing*, *classic* $\}$. Of course, obtaining such a heterogeneous cluster is not desirable. Though, in case the scoring function did not behave as expected, the dictionary generation method has to handle such heterogeneity. To find fix parts, we first look for letters that are in every words of $\mathcal{W}_g$. We call *message types* such fix letters — `c`, `l`, `a` and `i` in the example. For the word *clarity*, we can immediately deduce the variable parts. Though, for the words *declaration*, *classification* and *unclarified* multiple occurrences of the message types `c`, `i` and `a` appear and we need to know which one is a real fix part. The following table census the frequencies of the fix parts in each words.

	clarity	*claimed*	*cleansing*	*classical*	*unclarified*	*declaration*	*classification*	min.
`c`	1	1	1	1	1	1	2	1
`l`	1	1	1	1	1	1	1	1
`a`	1	1	1	1	1	2	2	1
`i`	1	1	1	1	2	1	3	1

We define the *perfect examples* as the words with the minimum frequencies of each message type — the first four words of the table. We aim to chose, for the words *unclarified*, *declaration* and *classification*, the occurrences of the message types that are fix parts. We call α_i the i^{th} occurrence of letter α in a word : in *unclarified*, $\texttt{i}_1$ is the `i` between the `r` and the `f`. To decide which occurrences of the message types to keep, we first look at the order in which the fix parts appear. Based on our perfect examples, the order is always $\texttt{c}, \texttt{l}, \texttt{a}, \texttt{i}$. This helps us chosing the c_1 occurrence of *classification* rather than c_2. However, it does not promote any of the `a` and `i` occurrences in the three problematic words. We therefore add another feature to the selection : the gap between the fix parts for each perfect example. This value cannot be calculated for the other words, since we would not know which occurrence to chose.

gap lengths	*clarity*	*claimed*	*cleansing*	*classical*	mean	interquartile range
`c - l`	1	1	1	1	1	0
`l - a`	1	1	2	1	1.25	0.25
`a - i`	2	1	3	2	2	0.5

The mean of the gaps can be seen as the ideal gaps that should separate the fix parts. For *unclarified*, the gap between `a` and $\texttt{i}_1$ is 2, and between `a` and $\texttt{i}_2$ is 4. Therefore, we successfully select the $\texttt{i}_1$ occurrence. Moreover, for *classification* we have the following gaps between occurrences of `a` and `i` :

`a`-occurrence	`i`-occurrence	occurences gap	expected gap	difference with expected
$\texttt{a}_1$	$\texttt{i}_1$	3	2	1
$\texttt{a}_1$	$\texttt{i}_2$	5	2	3
$\texttt{a}_1$	$\texttt{i}_3$	9	2	7
$\texttt{a}_2$	$\texttt{i}_3$	2	2	0

However, the huge gap between a_2 and l will fortunately have us chose a_1 — since we take the mean of the gaps. Nevertheless, it not sufficient for the word *declaration* :

a-occurrence	observed		difference with expected		mean difference
	l - a	a - i	l - a	a - i	
a_1	1	4	0	2	1
a_2	3	2	2	0	1

Indeed, the two occurrences of a are assigned the same score. We therefore add a last feature to our decision method : a measure of gap stability. In the previous example, we observe that the gap between the letters l and a has more stable values than the gap between a and i. Hence, the difference between the real and the ideal values of the gap should take into account the reliability of the gap. Thus, we propose to calculate the interquartile range of each gap (last column of table 2). This interquartile is inversely proportional to the coefficient attributed to the gaps. In our example, since the relation l − a is quite stable, it will be highly penalized for a_2 to have an extra gap of 2. The a_2 occurrence will finally be chosen as a fix part. With all the defined process of fix part selection, we simply deduce the variable parts as the variable letters and the unselected common letters. We obtain, in our example, the wanted result :

$$\left\{\begin{array}{l} \text{un} \\ \text{de} \\ \_ \end{array}\right. \text{cl} \left\{\begin{array}{l} \text{e} \\ \_ \end{array}\right. \text{a} \left\{\begin{array}{l} \text{r} \\ \text{ns} \\ \text{ss} \\ \text{arat} \\ \_ \end{array}\right. \text{i} \left\{\begin{array}{l} \text{ty} \\ \text{med} \\ \text{ng} \\ \text{cal} \\ \text{fied} \\ \text{on} \\ \text{fication} \end{array}\right.$$

The stemmed extracted from this group could then be _cl_a_i_. In practice, we hope not to obtain such an heterogeneous group. In a well-formed group we can hope to have more intelligible stemmers like _classif_ or _clar_. This new version of the generated stem could enhance string comparison results, which might be needed in NLP tasks. Moreover, dictionary look-ups will be easier for the user.

4. Evaluation

We propose to assess the results of our implementation of the RFreeStem stemmer. We aim to demonstrate that our stemming method offers better accuracy results than the traditionally used Porter stemmer, while proposing a better compression power. To do so, we propose to run our stemmer and Porter's on raw data before the application of a NLP task.

4.1. Evaluation framework

Our evaluation is divided in two discussions : the efficiency of our stemmer in two different text mining tasks in English, and the comparison of this efficiency depending

on the language of the corpus. For each discussion, we will compare the different versions of our algorithm — due to the possible values of the depth of the cut — with both Porter's and the unstemmed data.

Task 1: Text categorization is a popular text mining task that aims at automatically determining the class of a document within a corpus. To test the efficiency of our stemmer as a preprocessing step for text categorization, we implemented our method on the AustLII case report corpus [2]. We retrieved 6480 citations out of 717 different XML files. Those files contain both the text of the legal citation and a manually labelled category, among the following :

cited	referred to	applied	followed	considered	discussed	Others(12)
2847	1372	715	529	361	323	170

Our dataset is highly imbalanced since 43.9% of the citations have the label *cited*, whereas half of the classes only contains 1.25% of the data. Imbalanced datasets are reputed to be harder to process, so it will be interesting to see if our stemmer can help in that matter.

Task 2: Sentiment analysis has become a very popular task, in particular with the high expansion of social networks and their textual data. The aim is to evaluate sentiments in human written comments or reviews. To evaluate the ability of RFreeStem to can improve the results of this complex task, we chose the commonly used movie review dataset [3]. Firstly mentioned in (Pang, Lee, 2004), the corpus is still popularly used today (Ba *et al*., 2016), (Shen *et al*., 2018). It contains 10662 perfectly balanced reviews of movies, with 18196 different terms.

Since the state-of-the-art seems to agree that stemming is more adapted to inflectional languages, we propose to compare our stemming on related datasets from different languages. We therefore studied the amazon reviews in French and German[4] for sentiment analysis task (**Task 2**). Among the large amount of available data, we chose, for each language, a sample of 2000 reviews, respectfully of the proportion of positive and negative labels. Since the products are evaluated with a rating from 1 to 5, we extracted the binary labels with the following simple transformation rule : if the rating is strictly over 3, assign to positive class. Otherwise, assign to negative. We obtain the following datasets:

Language	#reviews	Positive values		Negative values		#features
French	2000	1634	81.7%	366	18.3%	11 583
German	2000	1704	85.2%	296	14.8%	16 143

2. http://www.austlii.edu.au/ (Galgani *et al*., 2012) uses it for categorization
3. http://www.cs.cornell.edu/people/pabo/movie-review-data/
4. https://s3.amazonaws.com/amazon-reviews-pds/readme.html

Similarly to the full datasets, the reviews are highly imbalanced, which allows us to test the robustness of our stemmer. We also note that German has more than 30% features more than French. Indeed, besides being highly inflective, German language tends to compose new words by juxtaposing two smaller ones. Moreover, it uses declensions to construct its articles and adjectives, which adds a large variety of word variants. German therefore seems like a perfect fit for stemming algorithms.

Finally, since our method is corpus-based, we claim that it is not language-dependant and could be applied on any language, especially the rarely documented ones, where it is hard to find a Porter-like stemmer. We propose to study a rare Roman Urdu data set[5] for a 3-class sentiment analysis (**Task 2**):

#reviews	Positive		Neutral		Negative		#features
20229	6013	29.8%	8929	44.1%	5287	26.1%	31888

The process we apply is as follow : for each dataset, we apply the different versions of our method to the whole raw data and therefore generate a stemming dictionary where each entry is a {word: stem} couple. As mentioned before, we want to observe the effect of the depth of the cut on the results. Therefore, we generate as many dictionary as we have candidate depths for the cut. Thanks to the study of (Harman, 1991), we have the idea of chosing to use 4-grams, that seem to be the most relevant, both on analytic and inflective languages.

We then run a supervised classification algorithm on different versions of the data : the raw data (pre-processed without stemming), the Porter data (pre-processed and stemmed with Porter stemmer) and all the d-RFreeStem data (pre-processed and stemmed with our algorithm, cut with a depth of d). We choose to train a Naive Bayesian classifier on 70% of the data, respecting the proportion of the labels. The 30% left are predicted with the trained classifier and the results are compared with their labels. To select the classifier input features, we simply calculate the document-term matrix, and sort the features in decreasing frequency order. For each configuration, we run several versions of the feature selection step. While i is less than $max(feature.frequencies)$, we select the features whose frequency is at least i. Each value of i gives a different number of features as input of the classifier. For each stemmer, we run several times all the features selection step to have a robust mean value.

4.2. Results and discussions

By running 20 times all the version of our algorithms, Porter's and the raw version, for all the possible feature selection configurations, we obtain the results presented in that subsection. As a measure of result improvement, we calculate the traditional F-measure, as well as the accuracy. We also evaluate the ICF value, to measure the

5. http://archive.ics.uci.edu/ml/datasets/Roman+Urdu+Data+Set, (Sharf, Rahman, 2018)

compression power of the understudied stemmer. Firstly, let us compare the accuracy

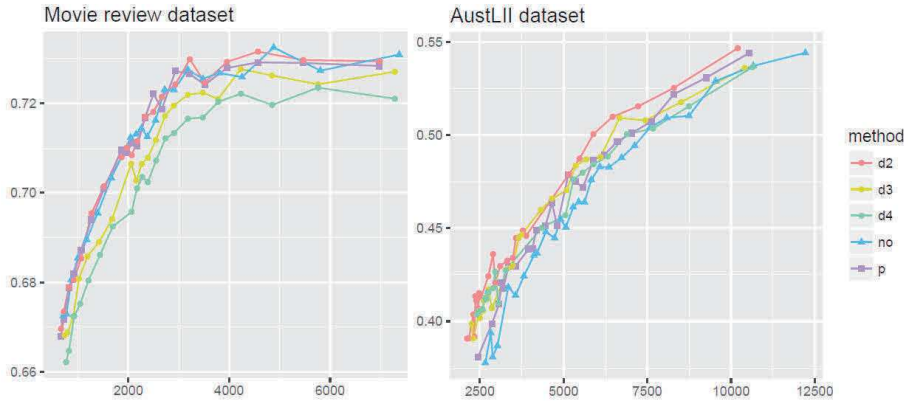

*Figure 3. Accuracy comparison for different #features between (**Task 2**) movie reviews binary classifier and (**Task 1**) AustLII multinomial classifier*

obtained in the two English mining tasks. We observe slighlty different results on those two evaluations. In the classification task, stemming systematically improves the results. The most aggressive versions of our algorithm 2-RFreeStem performs better than Porter, while 3-RFreeStem and Porter present similar results. Yet, the accuracy range is rather law : multiclass categorization seems to be a hard task to be applied on documents, and stemming can significantly help preserving a correct level of prediction. On the other hand, the sentiment analysis results are lukewarm. Almost all the stemming methods deteriorate the accuracy compared to the unstemmed version, apart from our most aggressive stemmer 2-RFreeStem. The movie review is also reputed to be a difficult task and it seems that stems might have masked the discrimination out of the variant forms of the words. Moreover, as explained before, better results are expected on inflective languages. To prove it, we compare this last result to the Amazon review ones.

Since French is a highly inflective language, each root has many variants and thus many words can be stemmed together while respecting their meanings. However, let us note that this corpus regroups human informal speeches. In informal speech, French tends to be less inflective : we would rather say *not happy* than *unhappy*. Nevertheless, our two most aggressive versions perform well and better than Porter (Figure 4). Similar results are observed with German texts, where 4 of our algorithms have at least one configuration where they outperform Porter, which struggles to reach the unstemmed performances. Actually, there is a clear segregation between Porter or the unstemmed version and our methods. This makes us think that our stemming algorithm is well adapted for German. Indeed, Porter stemming does not include any treatment for complex composed words, whereas (Braschler, Ripplinger, 2004) argues that decompounding task is an even more decisive factor than stemming for the German language. On the other side, our algorithm is theoretically capable of retrieving such complex linguistic structures, which can explains those encouraging results. Eventu-

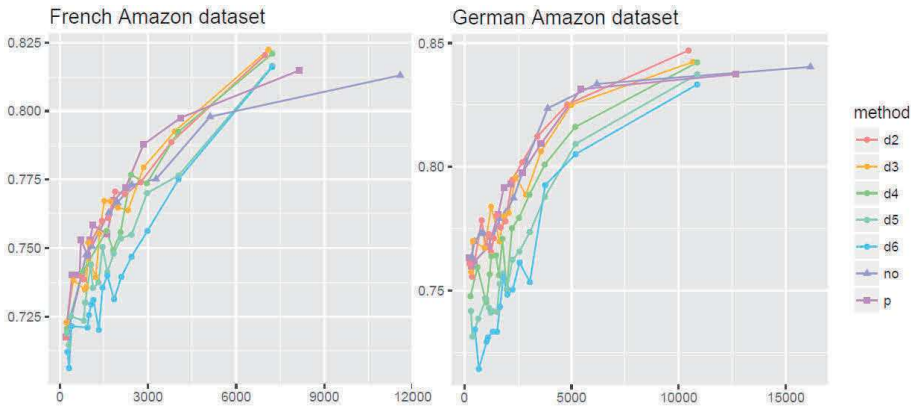

Figure 4. Accuracy comparison for different #features between French and German Amazon reviews dataset for a binary sentiment analysis

ally, those two highly inflective languages clearly show the compression power of our stemmer. Indeed, the maximum number of features for the unstemmed version is far greater than for our methods, due to the important vocabulary reduction performed. We naturally also detect a slight difference in that matter between the d-versions of our algorithms : the lower d value, the lower maximum number of features.

Finally, we propose to see how our method behaves on a poorly documented language as Urdu. For this language, we did not have a Porter version implemented and therefore only compared our results to the unstemmed one in Figure 5. We discover that only our most aggressive version can compete with the raw data. The improvement observed is far from being significant, but at least, we take satisfaction in proposing a method that will not deteriorate the accuracy for this left-behind language.

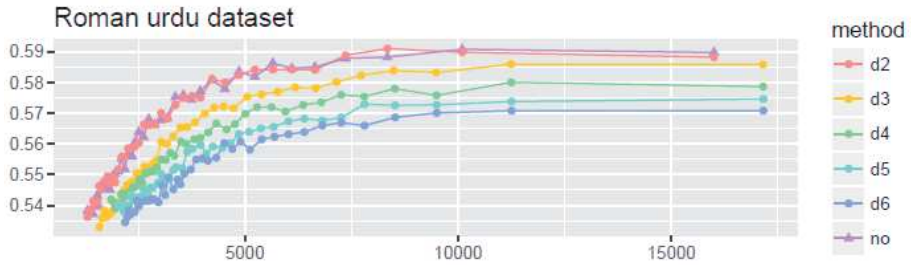

Figure 5. Comparison of the accuracy of the binary classifier prediction on Roman Urdu data for different #features

To conclude, the following table summarizes the evaluation features we wanted to observe: (i) F1 is the best F1 measure (over #features), (ii) is similarly the best accuracy value and ICF is the index compression factor of the method. For the English

version, we took the AustLII data since it is wider and shows more sensitivity to stemming. The following results are presented in percentage.

	English			French			German			Urdu		
	F1	Acc	ICF	F1	Acc	ICF	F1	Acc	ICF	F1	Acc	ICF
No	**33**.4	54.4	0	70.3	81.3	0	70.3	84.0	0	**59.1**	60.0	0
P	**33.4**	54.4	44.6	69.9	81.5	27.0	71.1	83.7	14.4	_	_	_
d2	33.0	**54.6**	**145**	**70.7**	82.0	**109**	71.2	**84.7**	**131**	**59.1**	**60.1**	**114**
d3	32.5	53.6	58.9	70.4	**82.2**	46.3	**71.5**	84.2	53.4	58.6	59.5	34.4

The best result are often obtained by our 2-RFreeStem version. This fact might be disappointing : in this version, only one clustering step is achieved, and no hierarchical tree is ever created. With this actually flat clustering, our method only consists in clustering the words depending on one of their 4-grams. A very similar method is proposed by (Pande *et al.*, 2013), yet they sort the ngrams very differently. However, our 3-RFreeStem version has sometimes proven more accurate (accuracy in French, F1-measure in German), and we argue that an even better solution could be found in refining our cutting algorithm. According to the results, for every groups created in the first splitting iteration, we could either chose to stop there or to split one more time, depending on an homogeneity criteria that is still to be defined.

5. Conclusion

Stemming is still systematically used as preprocessing step for text mining tasks. Since the sensitivity of stemming is highly language-dependant, we have proposed a multilanguage rule-free stemmer, based on corpus data. We have shown that it outperforms the famous Porter algorithm, and that its performance are even more spectacular on inflectional languages. An improvement of our method could be done by refining our dendrogram cutting method, thanks to a group homogeneity criteria. Moreover, instead of always working with 4-grams, we will soon try to enable the variation of the n length. We could also have proposed a wider evaluation framework by (i) analyzing the RFreeStem performance on agglutinative or rare languages (ii) assessing our stemmer accuracy improvement on Informational Retrieval, and even, since it is multilingual, on Cross Language Information Retrieval (iii) confronting our stemmer to other corpus-based methods, like (Soricut, Och, 2015). We let those interesting questions opened for a future contribution.

References

Adamson G. W., Boreham J. (1974). The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Information storage and retrieval*, Vol. 10, No. 7-8, pp. 253–260.

Ba J. L., Kiros J. R., Hinton G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Braschler M., Ripplinger B. (2004). How effective is stemming and decompounding for german text retrieval? *Information Retrieval*, Vol. 7, No. 3-4, pp. 291–316.

Dawson J. (1974). Suffix removal and word conflation. *ALLC bulletin*, Vol. 2, No. 3, pp. 33–46.

Frakes W. B., Baeza-Yates R. (1992). *Information retrieval: Data structures & algorithms* (Vol. 331). Prentice Hall Englewood Cliffs, NJ.

Galgani F., Compton P., Hoffmann A. (2012). Knowledge acquisition for categorization of legal case reports. In *Pacific rim knowledge acquisition workshop*, pp. 118–132.

Hafer M. A., Weiss S. F. (1974). Word segmentation by letter successor varieties. *Information storage and retrieval*, Vol. 10, No. 11-12, pp. 371–385.

Harman D. (1991). How effective is suffixing? *Journal of the american society for information science*, Vol. 42, No. 1, pp. 7–15.

Hull D. A. (1996). Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society for Information Science*, Vol. 47, No. 1, pp. 70–84.

Jivani A. G. *et al*. (2011). A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, Vol. 2, No. 6, pp. 1930–1938.

Kraaij W., Pohlmann R. (1994). Porter's stemming algorithm for dutch. *Informatiewetenschap*, pp. 167–180.

Kraaij W., Pohlmann R. (1996). Viewing stemming as recall enhancement. In *Sigir*, Vol. 96, pp. 40–48.

Krovetz R. (1993). Viewing morphology as an inference process. In *Proceedings of the 16th annual international acm sigir conference on research and development in information retrieval*, pp. 191–202.

Lovins J. B. (1968). Development of a stemming algorithm. *Mech. Translat. & Comp. Linguistics*, Vol. 11, No. 1-2, pp. 22–31.

Majumder P., Mitra M., Parui S. K., Kole G., Mitra P., Datta K. (2007). Yass: Yet another suffix stripper. *ACM transactions on information systems (TOIS)*, Vol. 25, No. 4, pp. 18.

Marcos-Pablos S., García-Peñalvo F. J. (2019). Information retrieval methodology for aiding scientific database search. *Soft Computing*, pp. 1–10.

Moral C., Antonio A. de, Imbert R., Ramírez J. (2014). A survey of stemming algorithms in information retrieval. *Information Research: An International Electronic Journal*, Vol. 19, No. 1, pp. n1.

Paice C. D. (1990, November). Another stemmer. *SIGIR Forum*, Vol. 24, No. 3, pp. 56–61. Retrieved from http://doi.acm.org/10.1145/101306.101310

Paice C. D. (1994). An evaluation method for stemming algorithms. In *Proceedings of the 17th annual international acm sigir conference on research and development in information retrieval*, pp. 42–50.

Paice C. D. (1996). Method for evaluation of stemming algorithms based on error counting. *Journal of the American Society for Information Science*, Vol. 47, No. 8, pp. 632–649.

Pande B. P., Tamta P., Dhami H. S. (2013). Generation, implementation and appraisal of an n-gram based stemming algorithm. *arXiv preprint arXiv:1312.4824*.

Pang B., Lee L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd annual meeting on association for computational linguistics*, p. 271.

Porter M. F. (1980). An algorithm for suffix stripping. *Program*, Vol. 14, No. 3, pp. 130–137.

Porter M. F. (2001). *Snowball: A language for stemming algorithms.* Retrieved from http://snowball.tartarus.org/

Sharf Z., Rahman S. U. (2018). Performing natural language processing on roman urdu datasets. *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, Vol. 18, No. 1, pp. 141–148.

Shen T., Zhou T., Long G., Jiang J., Pan S., Zhang C. (2018). Disan: Directional self-attention network for rnn/cnn-free language understanding. In *Thirty-second aaai conference on artificial intelligence.*

Sirsat S. R., Chavan V., Mahalle H. S. (2013). Strength and accuracy analysis of affix removal stemming algorithms. *International Journal of Computer Science and Information Technologies*, Vol. 4, No. 2, pp. 265–269.

Soricut R., Och F. (2015). Unsupervised morphology induction using word embeddings. In *Proceedings of the 2015 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 1627–1637.

Tala F. Z. (2003). A study of stemming effects on information retrieval in bahasa indonesia. *Institute for Logic, Language and Computation, Universiteit van Amsterdam, The Netherlands*.

Willett P. (2006). The porter stemming algorithm: then and now. *Program*, Vol. 40, No. 3, pp. 219–223.